I Rendu de monnaie

On suppose disposer de différentes pièces dont les valeurs sont stockées, dans l'ordre décroissant, dans une liste `Pieces`. On souhaite trouver le moyen de rendre une certaine somme `s` en utilisant le moins de pièces possibles.
On supposera que le stock de pièces disponible est infini (et donc on n'aura toujours toutes les pièces disponibles pour ce rendu de monnaie).
On supposera également que la plus petite pièce possède une valeur 1 ce qui fait que le rendu de monnaie sera toujours possible.

1. Approche gloutonne

L'approche gloutonne est définie ainsi : pour rendre la monnaie, on commence par rendre la pièce dont la valeur est la plus grande possible (et on recommence sur la somme restante)

1. Écrire une fonction récursive `glouton(s:int,P:list)->list` qui prend en argument la somme `s` à rembourser (supposée $\geqslant 1$) et la liste `P` des valeurs des pièces, et qui renvoie la liste des pièces à utiliser pour ce remboursement.
2. Que donne cette approche dans le cas où `Pieces = [10,5,2,1]` puis `Pieces = [30,24,12,6,3,1]` lorsque `s = 49`.

2. Programmation dynamique

3. Si on note N_s le nombre de pièces minimal nécessaire pour rembourser la somme s et $p_1, \dots, p_k$ la valeurs des pièces disponibles. Quelle est la valeur de N_s en fonction des termes $(N_i)_{i \leqslant s-1}$?
4. Écrire une fonction récursive `rendu(s:int,P:list)->int` qui prend en argument la somme `s` à rembourser (supposée $\geqslant 1$) et la liste `P` des valeurs des pièces, et qui renvoie le nombre de pièces minimal nécessaire pour rendre la somme `s`.
5. Rendre la fonction précédente plus efficace en utilisant la mémoïsation avec un dictionnaire.
6. Écrire une fonction `BasEnHaut(n:int,P:list)->int` qui effectue le même calcul que la fonction précédente mais en allant de bas en haut (programmation itérative) : on remplira une liste `L` de sorte que `L[i]` soit le nombre de pièces à utiliser pour rendre la somme `i`.
7. Évaluer la complexité de cette dernière fonction.

II Suite de Syracuse

La suite de Syracuse est définie par

$$u_0 = x \in \mathbb{N}^* \quad \text{et} \quad \forall n \in \mathbb{N}, u_{n+1} = \begin{cases} \dfrac{1}{2}u_n & \text{si } n \text{ est pair} \\ 3u_n + 1 & \text{sinon} \end{cases}$$

On conjecture que pour tout entier naturel $x \geqslant 1$, il existe un rang n_0 pour lequel $u_{n_0} = 1$; la suite devient alors 3-périodique (et prend successivement les valeurs 1,4,2).

1. Écrire une fonction récursive `Syracuse(x:int,n:int)->int` qui renvoie la valeur du $n^{\text{ème}}$ terme de la suite de premier terme `x`.

On appelle temps de vol en x, la valeur du plus petit entier n tel que la suite de Syracuse de premier terme x prenne la valeur 1.

2. Écrire une fonction `tempsVol(x:int)->int` qui renvoie le temps de vol en x.
3. En déduire une fonction `tempsVolMax(N:int)->int` qui renvoie le temps de vol maximal pour tous les entiers $x \leqslant N$.
4. Rendre cette fonction plus efficace en utlisant la technique de mémoïsation avec un dictionnaire.
5. Pourquoi est-il difficile d'écrire une fonction itérative, de bas en haut, qui renvoie la même valeur que `tempsVolMax(N)` ?